A Correct Abstract Machine for the Stochastic Pi-calculus

Andrew Phillips     Luca Cardelli

August 30th 2004
Microsoft Research
7 JJ Thomson Avenue
Cambridge, UK

Andrew Phillips - BioConcur 2004
Introduction

➢ Ongoing Experiment:
  ❑ Use process calculi to model biological systems

➢ Features of process calculi:
  ❑ Compositional modelling, analysis and simulation of systems.

➢ Potential Benefits:
  ❑ Understand complex systems by decomposing them into simpler subsystems.
  ❑ Analyse properties of subsystems using established theory.
  ❑ Predict behaviour of subsystems by running stochastic simulations.
  ❑ Predict properties and behaviour of composed systems.

➢ Pi-calculus: one of the simplest and most well-studied calculi.
Outline

➢ Graphical Pi-Calculus

➢ Gene Regulation by Positive Feedback [Priami et al., 2001]

➢ Abstract Machine for Stochastic Pi-Calculus

➢ Simulator for Stochastic Pi-Calculus
Stochastic Pi-Calculus

➢ Syntax:

\[ P, Q ::= \nu x P \quad \text{Restriction} \quad \Sigma ::= 0 \quad \text{Null} \]
\[ | \quad P | Q \quad \text{Parallel} \quad | \quad \pi.P + \Sigma \quad \text{Action} \]
\[ | \quad \Sigma \quad \text{Summation} \quad \pi ::= x\langle n \rangle \quad \text{Output} \]
\[ | \quad !\pi.P \quad \text{Replication} \quad | \quad x(m) \quad \text{Input} \]

➢ Semantics:

\[(x\langle n \rangle.P + \Sigma) \mid (x(m).Q + \Sigma') \quad \frac{\text{rate}(x)}{\rightarrow} \quad P \mid Q_{\{n/m\}}\]
\[P \xrightarrow{r} P' \quad \Rightarrow \quad P \mid Q \xrightarrow{r} P' \mid Q\]
\[P \xrightarrow{r} P' \quad \Rightarrow \quad \nu x P \xrightarrow{r} \nu x P'\]
\[Q \equiv P \land P \xrightarrow{r} P' \land P' \equiv Q' \quad \Rightarrow \quad Q \xrightarrow{r} Q'\]
Graphical Pi-Calculus

➢ An intuitive representation for pi-calculus. Like FSMs...

➢ But with all the features of pi: compositionality, restriction, communication, replication.

➢ Should be a 1-1 correspondence between graphics and text

➢ NO NEW THEORY
Graphical Syntax

$P, Q ::= \text{new } n \quad P$

Restriction

$\Sigma ::= \quad \Box$

Null

Parallel

$\Sigma \quad P$

Action

$\Sigma$ Summation

$\pi ::= x^{<n>}$ Output

$! P$

Replication

$x(m)$ Input
Graphical Semantics: Restriction

➢ Restriction creates a fresh name inside a given process.
The name $n$ is replaced with a fresh name $n_1$ that is unknown to $Q$. 
Graphical Semantics: Communication

➢ Two parallel summations can interact on a common channel.
➢ An output $x(n)$ can send a message $n$ on channel $x$ to an input $x(m)$. 
Graphical Semantics: Communication

- Message $n$ is assigned to $m$ in process $P'$.
➢ A replicated input can spawn a clone of a process.
Graphical Semantics: Replication

An output $x\langle n \rangle$ can send a message $n$ to a replicated input $!x(m)$. 
➢ A clone of $P$ is spawned and message $n$ is assigned to $m$ in the clone.
Trees vs Graphs

➢ By definition, a graphical pi process is a tree of nodes:

➢ Links between nodes in the tree can be encoded to represent recursive processes:

➢ The result is an arbitrary graph with two kinds of edges.
Ionization: $Na + Cl \rightleftharpoons Na^+ + Cl^-$

➢ $Na$ can ionize $Cl$ by sending its electron, with rate $100s^{-1}$

➢ $Cl^-$ can deionize $Na^+$ by sending its electron, with rate $10s^{-1}$

➢ State names are merely *annotations*
Ionization: \( Na + Cl \rightleftharpoons Na^+ + Cl^- \)

➢ *Na* can ionize *Cl* by sending its electron on the *ionize* channel
**Ionization:** $Na + Cl \rightleftharpoons Na^+ + Cl^-$

- $Na^+$ is positively charged and $Cl^-$ is negatively charged
Ionization: \( Na + Cl \rightleftharpoons Na^+ + Cl^- \)

➢ \( Cl^- \) can deionize \( Na^+ \) by sending its electron on the deionize channel
Ionization: $Na + Cl \rightleftharpoons Na^+ + Cl^-$

➢ $Na$ and $Cl$ are no longer charged
Ionization: $Na + Cl \rightleftharpoons Na^+ + Cl^-$
Ionization: $Na + Cl \rightleftharpoons Na^+ + Cl^-$
Ionization: $Na + Cl \rightleftharpoons Na^+ + Cl^-$
**Ionization:** $Na + Cl \rightleftharpoons Na^+ + Cl^-$
Ionization: $Na + Cl \rightleftharpoons Na^+ + Cl^-$
Ionization: \( Na + Cl \rightleftharpoons Na^+ + Cl^- \)
Ionization: $Na + Cl \rightleftharpoons Na^+ + Cl^-$
Virtual Experiment: \( \text{Na} + \text{Cl} \rightleftharpoons \text{Na}^+ + \text{Cl}^- \)
Covalent Bonding: $H + Cl \rightleftharpoons HCl$

➢ $H$ has a *private* electron.

➢ $H$ can share its electron with $Cl$ to form a covalent bond with rate $100s^{-1}$

➢ $HCl$ can break its private bond with rate $10s^{-1}$
Covalent Bonding: $H + Cl \rightleftharpoons HCl$

➢ $H$ has a private electron $e1$ that is not accessible from outside.
Covalent Bonding: $H + Cl \rightleftharpoons HCl$

➢ $H$ can share its electron with $Cl$ on the $share$ channel.
Covalent Bonding: $H + Cl \rightleftharpoons HCl$

➢ $H$ and $Cl$ share a private electron, to form $HCl$. 
Covalent Bonding: $H + Cl \rightleftharpoons HCl$

$\Rightarrow$ $HCl$ can break its private bond by synchronising on channel $e1$. 
Covalent Bonding: $H + Cl \rightleftharpoons HCl$

➢ $H$ and $Cl$ are no longer bound
Covalent Bonding: $H + Cl \rightleftharpoons HCl$
Covalent Bonding: \[ H + Cl \rightleftharpoons HCl \]
Covalent Bonding: \[ H + Cl \rightleftharpoons HCl \]
Covalent Bonding: $H + Cl \rightleftharpoons HCl$
Covalent Bonding: $H + Cl \rightleftharpoons HCl$
Covalent Bonding: $H + Cl \rightleftharpoons HCl$
Covalent Bonding: $H + Cl \rightleftharpoons HCl$
Covalent Bonding: \( H + Cl \rightleftharpoons HCl \)
Virtual Experiment: $H + Cl \rightleftharpoons HCl$
Gene Regulation by Positive Feedback

[Priami et al., 2001]
Gene Regulation by Positive Feedback

RNA_A
- rna_A()
- Bound_A
- TF
- Bound_TF
- Bound_aTF
- aTF
- bind<unbind, send, remove>
- new unbind
- new send
- new remove
- degrade()
- unbind<>
- bind<unbind, send, remove>
- remove<>
- A
- degrade()
- new unbind
- new send
- new remove
- protein_A()
- !Protein_A

Proteins
- protein_A<>
- translate()
- degrade’()
- rna_A()<
- transcribe() + degrade’<>
- + transcribe<>
- + translate<>
- !RNA_A

DNA_A
- transcribe() + tail()
- DNA_TF
- rna_TF<>
- transcribe’() + tail()
- !RNA_TF
- Protein_TF
- protein_TF()
- transcribe’() + translate()
- transcribe’() + translate<>
- protein_TF<>
- rna_TF<>
- transcribe() + tail()
- DNA_TF
- protein_TF()
- !RNA_TF
Gene Regulation by Positive Feedback

![Diagram of gene regulation by positive feedback]

- RNA_A binds to Bound_A, which can then send a signal (tail) to trigger degradation of RNA_A.
- Bound_TF binds to DNA_TF, leading to transcription and translation of Protein_TF, which in turn binds to RNA_A.
- Protein_A is translated from RNA_A, leading to degradation of RNA_A.

Andrew Phillips - BioConcur 2004
Gene Regulation by Positive Feedback

[Diagram of gene regulation processes]

- RNA_A
- Bound_A
- send<tail>
- unbind<>
- bind<unbind, send, remove>
- remove<>
- A
- degrade()
- new unbind
- new send
- new remove
- Protein_A

- Protein_TF
- RNA_TF
- DNA_TF
- Transcribe
- Transcribe'
- degrade>
- transcribe>
- degrade'<>
- transcribe'<>
- !RNA_A
- DNA_A
- Protein_A
- RNA_TF
- Protein_TF
Gene Regulation by Positive Feedback

RNA_A

Bound_A

send<tail>

unbind<>

bind<unbind, send, remove>

remove<>

degrade()

protein_A()

!Protein_A

degrade()""""""

protein_TF<>

protein_A<>

transcribe()"""

tail()"""

degrade<>"""

degrade’<>"""

RNA_A<>

ranscribe’<>"""""""""

tail()"""""""""

DNA_A

 RNA_TF

DNA_TF

RNA_TF<>

degrade<>""""""

degrade’<>""""""

proteins

Andrew Phillips - BioConcur 2004
Gene Regulation by Positive Feedback
Gene Regulation by Positive Feedback

Bound_A send\langle tail\rangle

A degrade()

unbind<>

bind<unbind,send,remove>

protein_A()  

!Protein_A

degrade()

protein_A<>  

declare()

transcribe()  

rna_A<>  

degrade<>  

transcribe<>  

transcribe'<>

!RNA_A

DNA_A

Proteins

declare()  

degrade'<>

!Protein_TF

!RNA_TF

Andrew Phillips - BioConcur 2004
Gene Regulation by Positive Feedback

- RNA_A
- Bound_A
- TF
- Bound_TF
- Bound_aTF
- aTF
- bind<unbind,send,remove>
- new unbind
- new send
- new remove
- !RNA_A
- degrade()
- protein_A()
Gene Regulation by Positive Feedback

Andrew Phillips - BioConcur 2004
Gene Regulation by Positive Feedback

Andrew Phillips - BioConcur 2004
Gene Regulation by Positive Feedback

Andrew Phillips - BioConcur 2004
Gene Regulation by Positive Feedback
Gene Regulation by Positive Feedback

![Diagram of gene regulation by positive feedback](image_url)
Gene Regulation by Positive Feedback

[Diagram of gene regulation processes involving RNA, proteins, and DNA]
Gene Regulation by Positive Feedback
Gene Regulation by Positive Feedback

Bound_A send<tail>

degrade()
unbind<> 
bind<unbind,send,remove>
remove<>  
protein_A() 

!Protein_A

degrade'()
protein_A<> 
translate() 

!RNA_A

degrade'<>
RNA_A<> 
transcribe'<>

!RNA_TF

degrade'
RNA_TF<> 
transcribe()

!Protein_TF

degrade()' 
protein_TF<> 
translate()
Gene Regulation by Positive Feedback
Gene Regulation by Positive Feedback

RNA_A

A

Bound_A

TF

Bound_TF

Bound_aTF

aTF

bind<unbind,send,remove>

new unbind

new send

new remove

protein_A()

!Protein_A

protein_A<>

translate()

degrade'() RNA_A
rna_A()

!RNA_A

DNA_A

tail() transcribe'() rna_A<> + degrade'<> + transcribe<> + translate<>

Proteins

dna_TF

RNA_TF
degrade'()

degrade()

degrade'() RNA_TF
degrade'()

degrade()

degrade'() RNA_TF
degrade'() RNA_TF

Bound_aTF

remove()

unbind() send(tail)
degrade() Bound_aTF

protein_TF>

protein_TF() protein_TF<>

!Protein_TF

DNA_TF

tail() transcribe'() rna_TF<> + transcribe<> + translate<>

Proteins

dna_TF

RNA_TF
degrade'()

degrade()

degrade() RNA_TF

degrade'() RNA_TF

Bound_aTF

remove() unbind() send(tail)
degrade() Bound_aTF

protein_TF>

protein_TF() protein_TF<>

!Protein_TF

DNA_TF

tail() transcribe'() rna_TF<> + transcribe<> + translate<>

Proteins

dna_TF

RNA_TF
degrade'()

degrade()

degrade() RNA_TF

degrade'() RNA_TF

Bound_aTF

remove() unbind() send(tail)
degrade() Bound_aTF

protein_TF>

protein_TF() protein_TF<>

!Protein_TF

DNA_TF

tail() transcribe'() rna_TF<> + transcribe<> + translate<>
Gene Regulation by Positive Feedback

Bound_A send<tail>
degrade()
unbind>
bind<unbind, send, remove>
remove>
new unbind
new send
new remove
protein_A()

!Protein_A

protein_A>
translate()
degrade’()
RNA_A

!RNA_A
degrade’()
transcribe’() 
rna_A>
transcribe()
tail() + translate’>

Proteins
dna_A>
transcribe()
tail()

!DNA_A

rna_A>
transcribe’() + transcribe’>
+ translate’>

!RNA_TF
degrade’()
translate()
protein_TF>

!Protein_TF
degrade’()
translate()
protein_TF>

!Protein_TF
degrade’()
translate()
protein_TF>

!Protein_TF
degrade’()
translate()
protein_TF>

!Protein_TF
degrade’()
translate()
Gene Regulation by Positive Feedback

Andrew Phillips - BioConcur 2004
Gene Regulation by Positive Feedback

RNA_A

RNA_TF

DNA_A

Proteins

Andrew Phillips - BioConcur 2004
Gene Regulation by Positive Feedback

[Diagram showing gene regulation processes involving RNA and proteins, with states and transitions labeled for unbind, send, remove, bind, degrade, and transcribe.]
Gene Regulation by Positive Feedback
Gene Regulation by Positive Feedback

[Diagram of gene regulation by positive feedback with labels and arrows indicating interactions between RNA, protein, DNA, and other components.]

Andrew Phillips - BioConcur 2004
Gene Regulation by Positive Feedback
Gene Regulation by Positive Feedback

Andrew Phillips - BioConcur 2004
Gene Regulation by Positive Feedback

![Diagram of Gene Regulation by Positive Feedback]

Andrew Phillips - BioConcur 2004
Gene Regulation by Positive Feedback

**Diagram:**
- **Bound_A** sends tail
  - degrade()
  - unbind<>
  - bind(unbind, send, remove)
  - remove<>
  - new unbind
  - new send
  - new remove

- **protein_A()**
  - !Protein_A

- **translate()**
- **degrade’()**
  - RNA_A

- **RNA_A**
  - rna_A()
  - !RNA_A

**Proteins:**
- **Bound_aTF**
  - unbind()
  - tail<> aTF
  - send(tail)
  - degrade()

- **Bound_TF**
  - unbind()
  - remove()

- **TF**
  - bind(unbind, send, remove)

**Actions:**
- **bind(unbind, send, remove)**
- **unbind<>**
- **send<>**
- **degrade’()**
- **degrade()**
- **transcribe<>**
- **transcribe’<>**
- **translate<>**
Gene Regulation by Positive Feedback

[Diagram of gene regulation with various pathways and reactions such as bind, unbind, send, remove, degrade, and transcribe.]
Abstract Machine for Stochastic Pi-Calculus (SPiM)

➢ Formalise how the simulator works (program specification).

➢ Prove properties about the simulator (program verification).

➢ Give greater confidence in the simulation results.

➢ Improve on existing simulators (BioSpi).
Machine Data Structures

➢ General Machine Term:

\[ \nu_{n_1} \nu_{n_2} \ldots \nu_{n_N} (\Sigma_1 :: \Sigma_2 :: \ldots :: \Sigma_M :: []) \]

➢ Syntax Definition:

\[
V, U ::= \nu n V \quad \text{Restriction} \\
| A \quad \text{List} \\
A, B ::= [] \quad \text{Empty} \\
| \Sigma :: A \quad \text{Summation}
\]
Machine Encoding

Encoding \([P]\):

\[ (P) \triangleq P \circ [] \]

Construction \((P \circ V)\):

\[ n \notin \text{fn}(P) \Rightarrow P \circ (\nu n V) \triangleq \nu n (P \circ V) \]

\[ 0 \circ A \triangleq A \]

\[ (P | Q) \circ A \triangleq P \circ Q \circ A \]

\[ n \notin \text{fn}(P \circ A) \Rightarrow (\nu m P) \circ A \triangleq \nu n (P_{\{n/m\}} \circ A) \]

\[ !\pi.P \circ A \triangleq (\pi.(P | !\pi.P) + 0) \circ A \]

\[ (\pi.P + \Sigma) \circ A \triangleq (\pi.P + \Sigma) :: A \]
Machine Execution

➢ Reduction \((V \xrightarrow{r} V')\):

\[
\begin{align*}
V \xrightarrow{r} V' & \Rightarrow \nu x V \xrightarrow{r} \nu x V' \\
\mid \quad x &= \text{Next}(A) \\
\land A &\succ (x(m).P + \Sigma)::A' \Rightarrow A \xrightarrow{\text{rate}(x)} P\{n/m\} \circ Q \circ A'' \\
\land A' &\succ (x\langle n\rangle.Q + \Sigma')::A''
\end{align*}
\]

➢ Selection \((A \succ B)\):

\[
\begin{align*}
A &\succ A \\
A &\succ \Sigma'::A' \Rightarrow \Sigma::A \succ \Sigma'::\Sigma::A' \\
\Sigma::A &\succ (\pi'.P' + \Sigma')::A \Rightarrow (\pi.P + \Sigma)::A \succ (\pi'.P' + \pi.P + \Sigma')::A
\end{align*}
\]
Channel Activity

➢ Choose next channel $x = \text{Next}(A)$ by stochastic algorithm [Gillespie, 1977]

➢ Gillespie chooses next channel based on *activity*:

\[
\text{activity} = \text{number of possible interactions on a channel}
\]

➢ In SPiM, activity of channel $x$ in term $A$:

\[
\text{Act}_x(A) = (\text{In}_x(A) \times \text{Out}_x(A)) - \text{Mix}_x(A)
\]

- $\text{In}_x(A)$ = number of unguarded *inputs* on channel $x$ in $A$.
- $\text{Out}_x(A)$ = number of unguarded *outputs* on channel $x$ in $A$.
- $\text{Mix}_x(A)$ = sum of $\text{In}_x(\Sigma_i) \times \text{Out}_x(\Sigma_i)$ for each summation $\Sigma_i$ in $A$. 

Andrew Phillips - BioConcur 2004
Gillespie: Choosing the Next Reaction $\text{Next}(A)$

1. For all $x \in fn(A)$ calculate $a_x = Act_x(A) \times \text{rate}(x)$

2. Store non-zero values of $a_x$ in a list $(x_{\mu}, a_{\mu})$, where $\mu \in 1...M$.

3. Calculate $a_0 = \sum_{\nu=0}^{M} a_{\nu}$

4. Generate two random numbers $n_1, n_2 \in [0, 1]$ and calculate $\tau, \mu$ such that:

   $$\tau = (1/a_0) \ln(1/n_1)$$

   $$\sum_{\nu=1}^{\mu-1} a_{\nu} < n_2 a_0 \leq \sum_{\nu=1}^{\mu} a_{\nu}$$

5. $\text{Next}(A) = x_{\mu}$ and $\text{Delay}(A) = \tau$. 
Correctness of the Machine

➢ Safety: no runtime errors (no crashes)

Lemma 1. \( \forall V. V \in \text{SPiM} \land V \xrightarrow{r} V' \Rightarrow V' \in \text{SPiM} \)

➢ Soundness: machine only performs valid executions steps (behaves well)

Theorem 1. \( \forall V. V \in \text{SPiM} \land V \xrightarrow{r} V' \Rightarrow [V] \xrightarrow{r} [V'] \)

➢ Completeness: machine can perform all execution steps of the calculus

Theorem 2. \( \forall P. P \in \text{SPi} \land P \xrightarrow{r} P' \Rightarrow [P] \xrightarrow{r} \equiv [P'] \).

➢ Termination: machine does not loop forever unnecessarily

Theorem 3. \( \forall P. P \in \text{SPi} \land P \not\xrightarrow{r} \Rightarrow [P] \not\xrightarrow{r} \)
Correctness of the Machine

➢ *Duration:* reactions in machine and calculus have same average duration

- Gillespie algorithm proved correct for selecting *next* reaction channel.
- Also need to ensure that reaction has correct *duration*
- E.g. reduction in $P_1$ is twice as fast as reduction in $P_2$:

$$
P_1 \triangleq x^n_r(P) + x^n_r(P) \parallel x^r_m.Q
\quad P_2 \triangleq x^n_r.P \parallel x^r_m.Q
$$

- Sufficient to ensure that machine and calculus have same channel activity.

**Proposition 1.**  $\forall V \in \text{SPiM}.\text{Act}_x(V) = \text{Act}_x(\llbracket V \rrbracket)$

**Proposition 2.**  $\forall P \in \text{SPi}.\text{Act}_x(P) = \text{Act}_x(\llbracket P \rrbracket)$
Implementation

➢ Abstract Machine maps almost directly to program code

➢ Implemented a polymorphic type system and type checker

➢ Correctness of the machine gives greater confidence in the simulation results

➢ Demo...
Conclusion

➢ Presented a graphical representation for pi-calculus:

- Precise, compositional, executable descriptions.
- Used to model regulatory systems at the molecular level.

➢ Presented an abstract machine for the stochastic pi-calculus:

- Correctness proof: safety, soundness, completeness, termination, duration.
- Maps readily to program code.
- Could be used as a basis for implementing new calculi.

➢ Built a simulator based on the machine.

- Plan to incorporate a graphical editor as a front-end.
References


Link Encoding

➢ Encoding uses restriction, replication, parallel composition and communication.

➢ A linked node → a replicated input on a fresh channel \( x \), in parallel with an output on \( x \).

➢ A link to the node → an output on \( x \).

➢ E.g.:
Safety Proof

Lemma 2. \( \forall V. V \in \text{SPiM} \land V \xrightarrow{r} V' \Rightarrow V' \in \text{SPiM} \)

Proof. By Lemma 3, Lemma 4 and by induction on reduction in \( \text{SPiM} \). \( \Box \)

Lemma 3. \( \forall A \in \text{SPiM}. A \succ B \Rightarrow B \in \text{SPiM} \)

Proof. By induction on selection in \( \text{SPiM} \). \( \Box \)

Lemma 4. \( \forall V. \forall P. V \in \text{SPiM} \land P \in \text{SPi} \Rightarrow P \circ V \in \text{SPiM} \)

Proof. By induction on construction in \( \text{SPiM} \). \( \Box \)
Soundness Proof

Theorem 4. \( \forall V. V \in \text{SPiM} \land V \xrightarrow{r} V' \Rightarrow [V] \xrightarrow{r} [V'] \)

Proof. By Lemma 5, Lemma 6 and by induction on reduction in \( \text{SPiM} \).

Lemma 5. \( \forall A. A \in \text{SPiM} \land A \succ B \Rightarrow [A] \equiv [B] \)

Proof. By induction on selection in \( \text{SPiM} \).

Lemma 6. \( \forall V. \forall P. V \in \text{SPiM} \land P \in \text{SPi} \Rightarrow [P \circ V] \equiv P \ | \ [V] \)

Proof. By induction on construction in \( \text{SPiM} \).
Completeness Proof

Theorem 5. \( \forall P. P \in \text{SPi} \land P \xrightarrow{r} P' \Rightarrow [P] \xrightarrow{r} \equiv [P'] \).

Proof. By Lemma 7 and by induction on reduction in SPi, where the rule for parallel composition is expanded over the remaining rules. \( \square \)

Lemma 7. \( P \equiv Q \Rightarrow [P] \equiv [Q] \)

Proof. By induction on structural congruence in SPi. \( \square \)

Lemma 8. \( \forall V. V \in \text{SPiM} \land U \equiv V \land V \xrightarrow{r} V' \Rightarrow \exists U'. U \xrightarrow{r} U' \land U' \equiv V' \)

Proof. By induction on structural congruence in SPiM. \( \square \)
Structural Congruence

\[ V \equiv_\alpha U \Rightarrow V \equiv U \]
\[ x \notin fn(V) \Rightarrow \nu x V \equiv V \]
\[ \nu x \nu y V \equiv \nu y \nu x V \]
\[ \Sigma :: \Sigma' :: A \equiv \Sigma' :: \Sigma :: A \]
\[ A \equiv A' \Rightarrow \Sigma :: A \equiv \Sigma :: A' \]
\[ (\pi.P + \pi'.P' + \Sigma) :: A \equiv (\pi'.P' + \pi.P + \Sigma) :: A \]
\[ \Sigma :: A \equiv \Sigma' :: A \Rightarrow (\pi.P + \Sigma) :: A \equiv (\pi.P + \Sigma') :: A \]
Termination Proof

Theorem 6. \( \forall P. P \in SPi \land P \not\rightarrow \Rightarrow [P] \not\rightarrow \)

Proof. By Theorem 4 and by basic relationships between encoding and decoding. \( \square \)
Graphical Semantics

➢ Requires some imagination: for substituting names and for cloning graphs.
➢ Output $x\langle n \rangle$ can send a message to input $x(m)$ on channel $x$. 
➢ $n$ is assigned to $m$ in process $P'$.
➢ Output $x\langle n\rangle$ can send a message to replicated input $!x(m)$.
➢ A clone of $P'$ is spawned and $n$ is assigned to $m$ in the clone of $P'$. 
Replicated output $!x^n$ can send a message to input $x^m$. 
➢ A clone of $P$ is spawned and $n$ is assigned to $m$ in $P'$.
Replicated output $!x(n)$ can send a message to replicated input $!x(m)$. 

Andrew Phillips - BioConcur 2004
➢ Clones of $P$ and $P'$ are spawned, and $n$ is assigned to $m$ in the clone of $P'$.
Ionization: $Mg + 2Cl \rightleftharpoons Mg^{+2} + 2Cl^-$